Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра «Вычислительная техника»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись
«_____»_____2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Унифицированный метод управления антенными постами
контрольно-измерительного комплекса космических аппаратов
09.04.01 — Информатика и вычислительная техника
09.04.01.04 — Технологии разработки программного обеспечения

Научный руководитель	_____ подпись, дата	ведущий инженер, канд.техн.наук	В.А. Хабаров
Выпускник	_____ подпись, дата		Д.В. Сафиуллин
Рецензент	_____ подпись, дата	ведущий инженер, аспирант	И.Н. Рыженко
Нормоконтролер	_____ подпись, дата	доцент, канд.техн.наук	А.И. Постников

Красноярск 2019

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра «Вычислительная техника»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись
«_____»_____20__ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме магистерской диссертации

Студенту Сафиуллину Дмитрию Валерьевичу.

Группа: КИ17-01-4м Направление (специальность): 09.04.01 Информатика и вычислительная техника.

Тема выпускной квалификационной работы: "Унифицированный метод управления антенными постами контрольно-измерительного комплекса космических аппаратов".

Утверждена приказом по университету _____ № _____

Руководитель ВКР: В.А. Хабаров, ведущий инженер, АО "НПП "Радиосвязь".

Исходные данные для ВКР: реализовать унифицированный метод управления антенными постами контрольно-измерительного комплекса космических аппаратов.

Перечень разделов ВКР: анализ задания на ВКР, разработка архитектуры системы, разработка программного обеспечения, проверка работоспособности системы.

Перечень графического материала: UML диграммы классов, презентация.

Руководитель ВКР _____ В.А. Хабаров

подпись

Задание принял к исполнению _____ Д.В. Сафиуллин

подпись

«_____»_____ 20__ г.

РЕФЕРАТ

Выпускная квалификационная работа по теме «Унифицированный метод управления антенными постами контрольно-измерительного комплекса космических аппаратов» содержит 50 страниц текстового документа, 27 рисунков, 6 использованных источник.

СИСТЕМА СВЯЗИ, РАДИОДОСТУП, QT, АНТЕННА, TCP, UDP, SNMP, CAN.

Объект работы: унифицированный метод управления антенными постами контрольно-измерительного комплекса космических аппаратов.

Задачи:

- выполнить анализ предметной области;
- разработать архитектуру программной части системы;
- разработать программное обеспечение, взаимодействующее с антенным постом;
- выполнить анализ полученных результатов.

Во введении раскрывается актуальность работы, ставятся цели и задачи.

В первой главе изложены результаты анализа существующих решений, а так же результаты выбора методов и средств разработки.

Во второй главе приведены результаты разработки архитектуры системы и ее программного обеспечения.

В третьей главе описан процесс практической реализации программной системы - её классов, а так же результаты разработки.

В четвёртой части изложены результаты проверки работы программы, указываются выявленные недостатки.

В результате работы создана реализация абстрактной антенны, с помощью которой можно взаимодействовать с различными видами антенн.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С момента ввода спутников в эксплуатацию, они стали незаменимой и неотъемлемой частью нашего современного высокотехнологичного мира. Они используются в самых разных сферах: спутники военного назначения, спутники связи, навигации и геодезии, а также спутники гражданского назначения. Первый в мире искусственный спутник земли (ИСЗ) был запущен в СССР 4 октября 1957 года[1]. В то время ресурсы необходимые для запуска спутника были только у государств, однако спустя время частные компании также смогли позволить себе это. Сейчас благодаря развитию нанотехнологий стали производить сверхмалые спутники формата кубсат, такой спутник весит от одного до нескольких килограмм и запуск спутника стал возможен для частных лиц. На данный момент в космос было отправлено более 6000 спутников[2].

Несмотря на огромное развитие в сфере космоса, разработка спутника до сих пор является не тривиальной задачей. Как и в любом другом производстве, важной частью создания спутника является его тестирование. Особенно это касается сферы безопасности и навигации. Если при разработке спутника гражданского назначения были допущены ошибки, которые в последствии не были устранены, то запуск такого спутника будет иметь в основном финансовые последствия. Для спутников военного назначения и навигации последствия будут куда более серьёзными. Поэтому тестированию должно проводиться по всем параметрам и возможностям спутника, а также сам процесс тестирования должен быть удобным и при возможности автоматизированным. Для этого создается контрольно-измерительный наземный комплекс (КИНК).

КИНК предназначен для автономного измерения и оценки параметров стволов и трактов бортовых ретрансляционных комплексов (БРТК) космических аппаратов (КА), автоматизированного приема и передачи тестовых сигналов на БРТК, автоматизированной настройки, контроля и отображения технического состояния аппаратуры, входящей в состав комплекса.

В состав КИНК входит контрольно-проверочная аппаратура (КПА), антенные посты (АП) с конвертирующей аппаратурой. Каждая связка КПА и АП предназначены для тестирования КА в установленном рабочем диапазоне частот: C-, X-, Ka/Q-диапазоне.

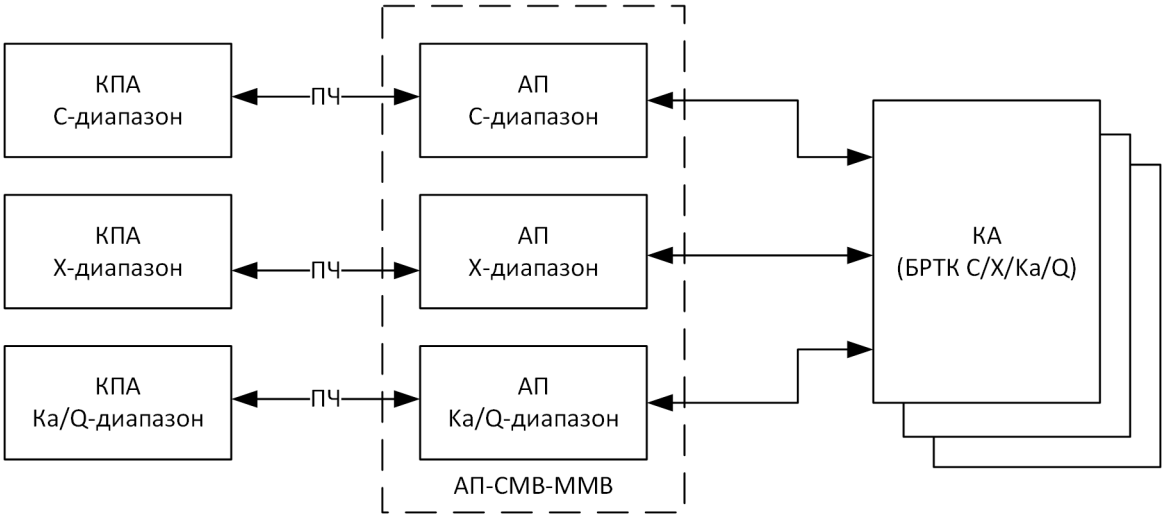

Рисунок 1 — Структурная схема КИНК

Основная сложность состоит в разработке программного обеспечения (ПО) для управления антеннами, так как интерфейс подключения, транспортный протокол, а также протокол управления антенной на каждом АП сильно отличается, а функции антенн в целом схожи. Решением данной проблемы будет разработка единой системы управления антеннами - абстрактной антенны.

Актуальность темы обусловлена необходимостью уменьшения трудозатрат и увеличения эффективности процесса разработки програм управления антеннами.

Цель работы: разработка ПО для управления антеннами через единообразный интерфейс.

Для достижения указанной цели в работе решаются следующие **задачи**:

- выполнение аналитического обзора задачи;
- разработка основных требований, архитектуры и алгоритмов для работы создаваемого ПО;
- разработка ПО на основе ранее разработанной архитектуры и алгоритмов;
- моделирование и тестирование разработанного ПО.

Основные положения, выносимые на защиту:

- метод проектирования абстрактной антенны;
- методы и алгоритмы взаимодействия программной и аппаратной ча-

сти системы.

Научная новизна: разработан метод проектирования и реализации абстрактной антенны позволяющий снизить трудозатраты и увеличить эффективность процесса разработки управления антеннами.

Практическая ценность данного метода заключается в том, что спроектировав и реализовав абстрактную антенну, можно получить единый интерфейс управления для антенн различающихся по назначению, типу подключения и протоколу обмена.

1 Анализ предметной области

Для разработки абстрактной антенны необходимо сравнить использующиеся в АП антенны. Выделив общий функционал станет возможным создать абстрактное представление, через которое будет возможность работать с любым из АП.

1.1 Антенный пост С-диапозон

Двухосная неортогональная антенна. Устанавливается на неподвижный наземный объект, подразумевает работу в неподвижном состоянии. Имеет следующий функционал: включение, выключение, вращение по азимуту на 359 градусов, вращение по углу-место ограничено в диапазоне от -45 до 90 градусов, ручное наведение, автоматическое наведение, в том числе по схемам организации связи (СХОС).
Интерфейс подключения: CAN
Транспортный протокол: COM

1.2 Антенный пост Х-диапозон

Двухосная ортогональная антенна. Устанавливается на наземной технике. Имеет возможность работы в движении. Имеет следующий функционал: включение, выключение, вращение по азимуту на 359 градусов, вращение по углу-место ограничено в диапазоне от 0 до 90 градусов, переход в транспортное положение, ручное наведение, автоматическое наведение, в том числе по СХОС запросам.
Интерфейс подключения: Ethernet
Транспортный протокол: TCP

1.3 Антенный пост Ka/Q-диапозон

Трёхосная антенна, устанавливается на морской технике. Имеет следующий функционал: включение, выключение, вращение по азимуту на 359

градусов, вращение по углу-место ограничено в диапазоне от 0 до 90 градусов, переход в транспортное положение, ручное наведение, автоматическое наведение, в том числе по СХОС запросам.
Интерфейс подключения: Ethernet
Транспортный протокол: UDP

1.4 Выводы

Антенные посты включают в себя антенны, обладающие схожим функционалом. Таким как: включение, выключение, вращение, наведение по целеуказаниям, сброс в начальное положение, обработка СХОС запросов. Это основной функционал которым необходимо наделить абстрактную антенну. Также из описания антенн видно, что управляются они посредством разных транспортных протоколов, таких как TCP, UDP и CAN. Чтобы ещё более упростить задачу разработки ПО, имеет смысл реализовать классы работающие с этими протоколами и расположить их в одной библиотеке. Такое решение позволит использовать эту библиотеку в других проектах и при необходимости расширять.

2 Моделирование системы

2.1 Общая архитектура проектов АП

На рисунке 2.1 видно что процесс разработки проекта состоит из реализации модели устройства, протокола обмена, а также адаптера для работы с сетевым интерфейсом.

Рисунок 2.1 — Структурное изображение архитектуры проекта АП

Абстрактная антенна и библиотека адаптеров, поможет снизить трудозатраты необходимые для разработки проекта.

Рисунок 2.2 — Структурное изображение архитектуры проектов АП с использование абстрактной антенны и библиотеки адаптеров

Qt предоставляет особый механизм коммуникации между объектами — систему сигналов и слотов. Сигнал вырабатывается, когда происходит определенное событие[3]. Слот это функция, которая вызывается в ответ на определенный сигнал. На этом механизме и будет построена взаимосвязь между объектами. Благодаря этому будет достигнут необходимый уровень абстракции.

2.2 Модель абстрактной антенны

Как видно на рисунке 2.3 абстрактная антенна будет состоять из отдельных модулей. Каждый модуль будет выполнять определённый функционал. Также в некоторых модулях будут реализованы сигналы и слоты, подключение к которым обеспечит связь между абстрактной антенной и реализацией протокола конкретной антенны. Все модули будут объединенны в классе-контроллере.

Список классов абстрактной антенны:

- AimSettings;
- AimAntenna;
- AimSpacecraft;
- AimSpacecraftGso;
- AimSpacecraftHeo;
- AimSpacecraftsData;
- AimDataGso;
- AimDataHeo;
- AimCalculation;
- AimRequest;
- AimRequestManager;
- AimController.

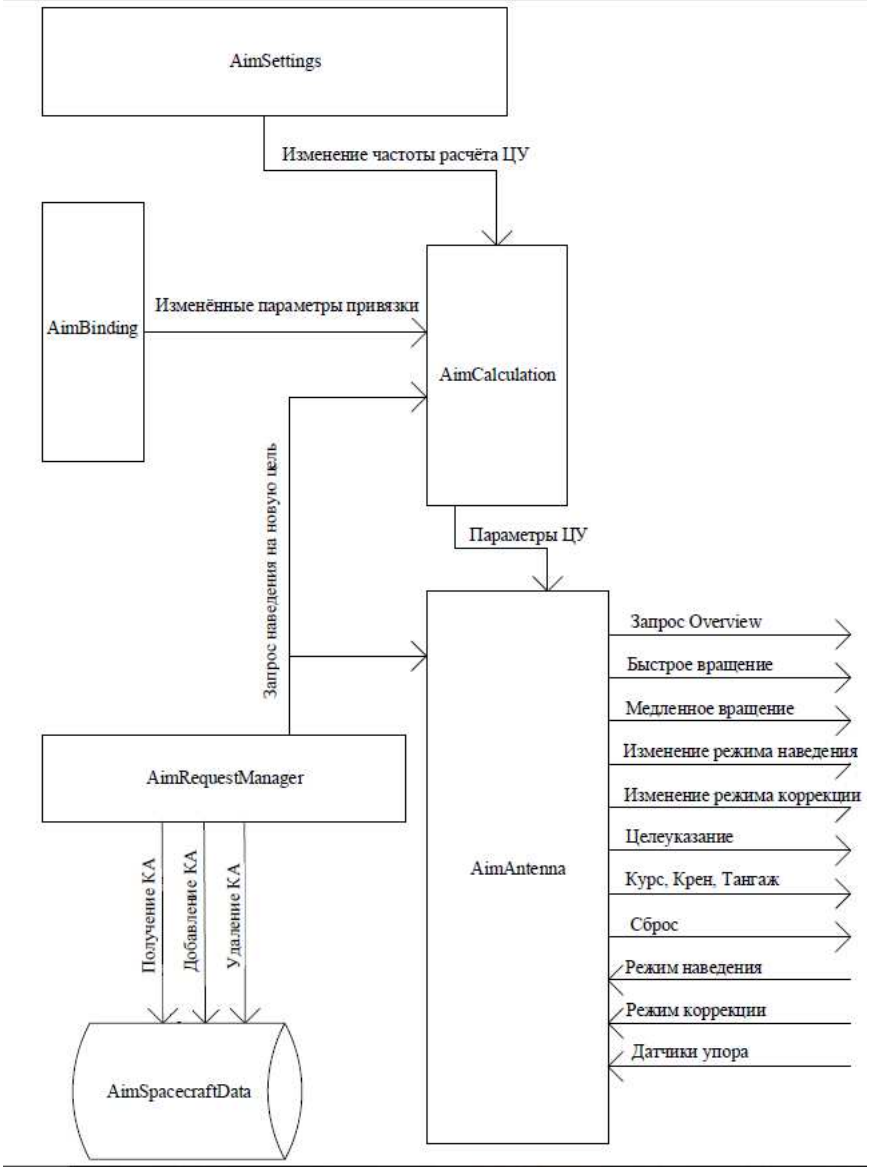

Рисунок 2.3 — Схема связей проекта абстрактная антенна

2.2.1 Организация класса AimSettings

Класс *AimSettings* содержит настройки необходимые для работы и управления проектом. Этот класс будет подключаться к другим модулям антенны.

Основные свойства класса:

- ***QString*** *m_systemName* — системное имя ;
- ***QString*** *m_debugName* — имя при отладке;
- ***bool*** *m_hasGso* — работает ли данная антенна с КА геостационарной орбиты (ГСО);
- ***bool*** *m_hasHeo* — работает ли данная антенна с КА высокой эллипти-

ческой орбиты (ВЭО);

- ***bool*** *m_hasRotating* — возможно ли ручное управление антенной;
- ***bool*** *m_hasTriming* — возможно ли проводить коррекцию наведения;
- ***bool*** *m_canHandleRequestByShos* — возможно ли управление наведением по средству СХОС запросов;
- ***bool*** *m_shouldRestoreLastRequest* — восстанавливать ли запрос, который был активен на момент закрытия программы;
- ***bool*** *m_canHandleRequestByShos* — возможно ли управление наведением по средству СХОС запросов;
- ***bool*** *m_hasAntenna* — возможно ли управление наведением по средству СХОС запросов;
- ***int*** *m_calculationFrequency* — частота расчёта целеуказаний (ЦУ);
- ***int*** *m_integrationInterval* — интервал интегрирования;
- ***bool*** *m_shouldCheckAntennaRegime* — синхронизировать ли режимы в реальном времени.

Один сигнал:

- ***void*** *calculationFrequencyChangedSignal ()* — частота расчёта ЦУ была изменена.

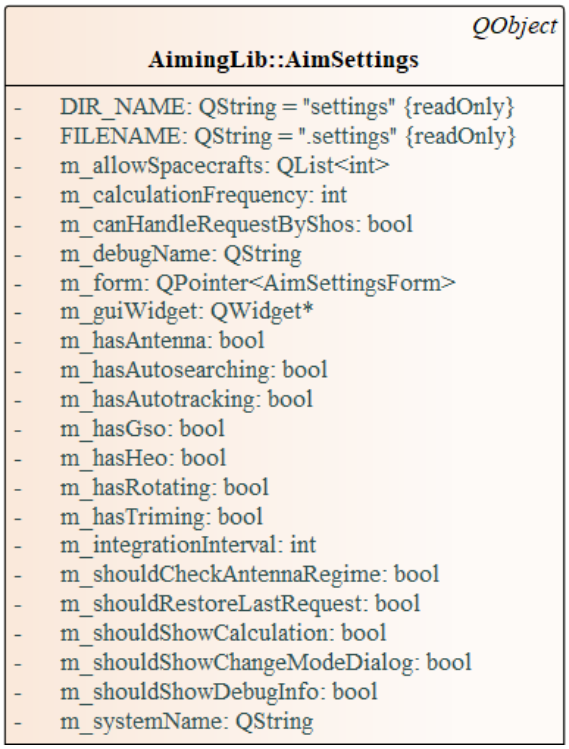

Рисунок 2.4 — UML диаграмма класса AimSettings

2.2.2 Организация класса AimAntenna

Управление антеннами будет производиться в классе AimAntenna. В нём будут реализованы сигналы отправляя которые мы будем иметь возможность контролировать положение антенны. Также в этом классе реализуем слоты для получения установленных режимов наведения и коррекции наведения, а также состояния датчиков упора.

В классе описываются три перечисляемых типа:

- ***enum class EnPointingMode*** — описание основных режимов первоначального наведения;

- ***enum class EnCorrectionMode*** — описание режимов коррекции наведения;

- ***enum class EnStopTrackerState*** — возможные значения датчиков упора.

Основные свойства класса:

- ***EnPointingMode*** *m_pointingMode* — выбранный режим первоначального наведения;

- ***EnCorrectionMode*** *m_correctionMode* — выбранный режим коррекции наведения;

- ***CCoordinate*** *m_azimuth* — целеуказание по азимуту;

- ***CCoordinate*** *m_elevation* — целеуказание по угол-месту;

- ***int*** *m_integrationInterval* — интервал интегрирования;

- ***bool*** *m_shouldCheckAntennaRegime* — синхронизировать ли режимы в реальном времени;

- ***EnStopTrackerState*** *m_stopTrackerTop* — значение верхнего датчика упора;

- ***EnStopTrackerState*** *m_stopTrackerBottom* — значение нижнего датчика упора;

- ***EnStopTrackerState*** *m_stopTrackerLeft* — значение левого датчика упора;

- ***EnStopTrackerState*** *m_stopTrackerRight* — значение правого датчика упора;

- ***bool*** *m_hasSignal* — наличие сигнала;

- ***int*** *m_signalLevel* — уровень сигнала;

- ***QList<AimSignalLevelSource*>*** *m_signalLevelSources* — источники сигнала;

- ***int*** *m_currentSignalLevelSource* — номер текущего источника сигнала.

Восемь сигналов для управления антенной:

- ***void*** *signalLevelSignal (**int** hasSignal , **double** signalLevel)* — источник и уровень сигнала;

- ***void*** *designationSignal (**int** azimuth, **int** elevation)* — установка целеуказаний;

- ***void*** *pointingModeSignal(**int** mode)* — установка режима наведения;

- ***void*** *correctionModeSignal (**int** mode)* — установка режима коррекции наведения;

- ***void*** *requestStateViewSignal (**QWidget** *widget, **QWidget** *window)* — запрос окна отображающие основные параметры антенны;

- ***void*** *resetSignal ()* — сброс положения антенны в начальное состояние;

- ***void*** *rotateSignal (**int** direction)* — ручное наведение антенны;

- ***void*** *trimSignal(**int** direction)* — ручная коррекция наведения антенны.

Четыре публичных слота:

- ***void*** *aimingSlot(**int** azimuth, **int** elevation)* — получение целеуказаний от модуля AimCalculation;

- ***void*** *stateModeSlot (**int** pointingMode, **int** correctionMode , **bool** isOnline)* — получения состояния антенны и текущих режимов работы;

- ***void*** *stateStopTrackersSlot (**int** top , **int** bottom, **int** left , **int** right)* — получение состояния стоп-датчиков антенны.

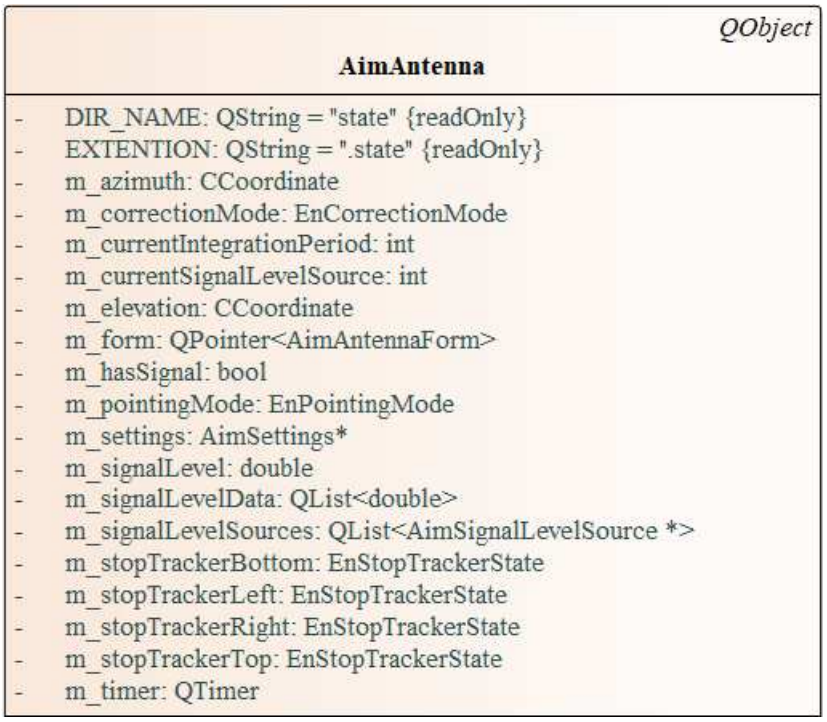

Рисунок 2.5 — UML диаграмма класса AimAntenna

2.2.3 Организация класса AimSpacecraft, AimSpacecraftGso и AimSpacecraftHeo

При работе с антенной будет выполняться наведение на КА. КА в нашем случае делятся на два типа ГЕО и ВЭО, в зависимости от орбиты на которой они будут работать.

Как видно из UML диграммы на рисунке 2.6, базовый класс AimSpacecraft описывает два перечисляемых типа:

- ***enum class EnOrbit*** — варианты орбит на которой КА будет использоваться;
- ***enum class EnType*** — описание возможных типов КА.

Также класс содержит следующие приватные свойства:

- ***EnOrbit*** *m_orbit* — орбита на которой работает КА;
- ***EnType*** *m_type* — тип КА;
- ***int*** *m_number* — номер КА.

Класс AimSpacecraftGso наследуется от AimSpacecraft и расширяется одним свойством:

- ***CCoordinate*** *Longitude* — долгота на которой расположен КА.

Рисунок 2.6 — UML диаграмма класса AimSpacecraft

Класс AimSpacecraftHeo наследуется от AimSpacecraft и расширяется следующими свойствами:

- ***unsigned char** *m_rawData* — массив данных с данными необходимыми для расчёта наведения;
- ***int*** *m_rawDataSize* — размер массива данных;
- ***int*** *m_day* — день для которого соответствуют данные ;

- ***int*** *m_month* — месяц для которого соответствуют данные.

Рисунок 2.7 — UML диаграмма класса AimSpacecraftHeo

2.2.4 Организация классов AimSpacecraftsData, AimDataGso и AimDataHeo

Также понадобится место для хранения объектов AimSpacecraft и предоставление к ним единой точки доступа. Таким местом будет AimSpacecrafts, который объединит в себе два класса-хранилища AimDataGso и AimDataHeo.

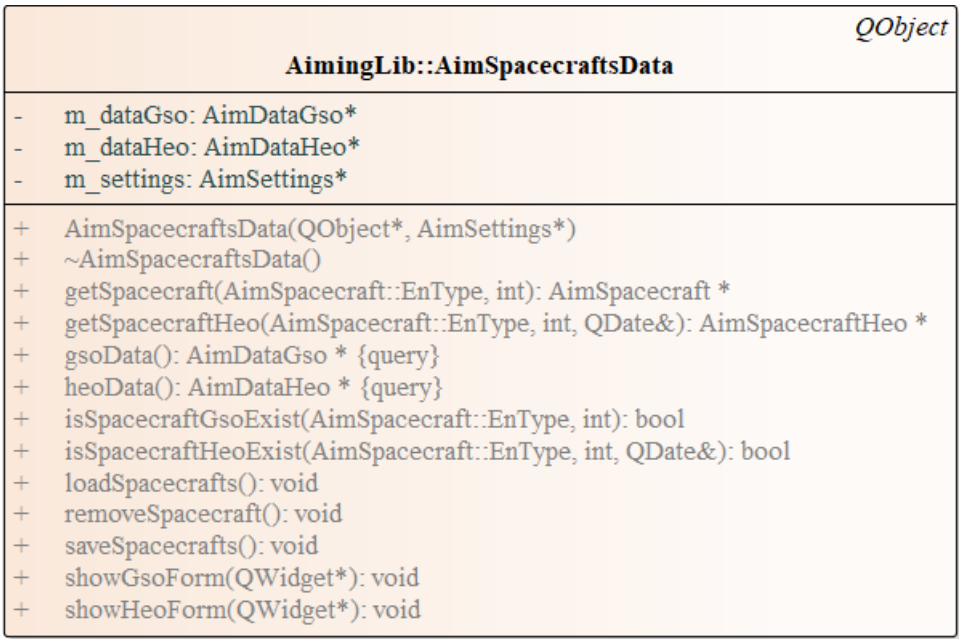

Рисунок 2.8 — UML диаграмма класса AimSpacecraftsData

Основные свойства класса AimDataGso:

- ***AimSettings*** **m_settings* — настройки абстрактной антенны;
- ***QString*** *m_dirName* — путь к директории с файлами с ГСО данными;
- ***QString*** *m_fileName* — имя файла с ГСО данными;
- ***QList<AimSpacecraftGso*>*** *m_spacecrafts* — список КА ГСО.

Основные свойства класса AimDataHeo:

- ***AimSettings*** **m_settings* — настройки абстрактной антенны;
- ***QString*** *m_directory* — директория с файлами данных;
- ***QTimer*** *m_timer* — таймер проверки директории;
- ***QList<AimSpacecraftHeo*>*** *m_spacecrafts* — список КА ВЭО.

Основные свойства класса AimSpacecraftsData:

- ***AimDataGso*** **m_dataGso* — указатель на хранилище КА ГСО;
- ***AimDataHeo*** **m_dataHeo* — указатель на хранилище КА ВЭО.

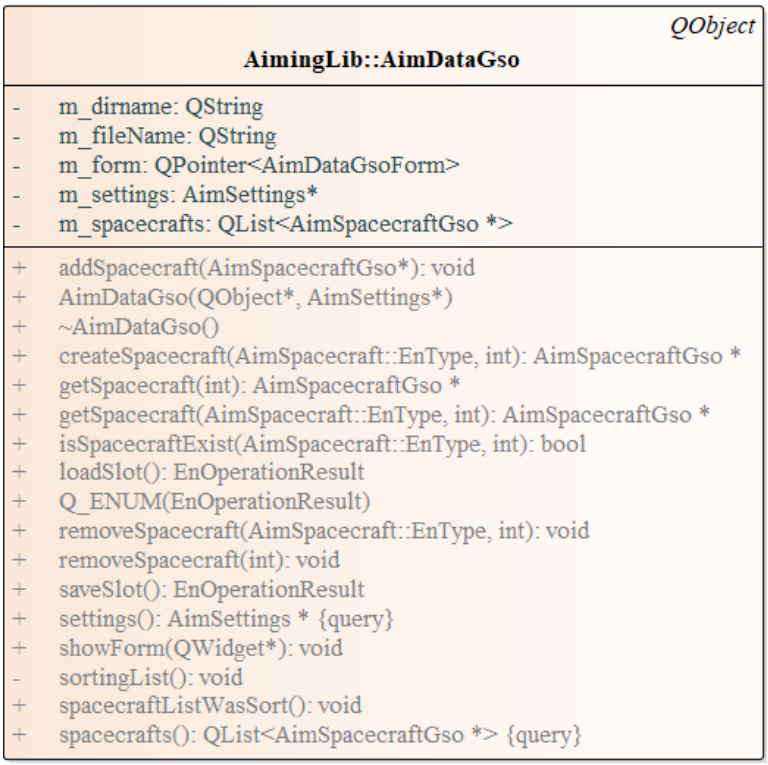

Рисунок 2.9 — UML диаграмма класса AimDataGso

Рисунок 2.10 — UML диаграмма класса AimDataHeo

2.2.5 Организация классов AimBinding

AimBinding класс отвечающий за привязку АП. С помощью него можно изменить источник данных широты, долготы и высоты, а также источник углов ориентации. Либо задать эти значения вручную.

Рисунок 2.11 — UML диаграмма класса AimBinding

Основные свойства класса:

- ***int*** *m_currentLocationSourceId* — текущий идентификатор источника широты, долготы и высоты;

- ***int*** *m_currentOrientationSourceId* — текущий идентификатор источника углов ориентации;

- ***CCoordinate*** *ManualLatitude* — введённое вручную значение широты;

- ***CCoordinate*** *ManualLongitude* — введённое вручную значение долготы;

- ***int*** *m_manualHeight* — введённое вручную значение высоты;

- ***CCoordinate*** *ManualVector* — введённое вручную значение курса;

- ***CCoordinate*** *ManualRoll* — введённое вручную значение крена;

- ***CCoordinate*** *ManualPitch* — введённое вручную значение тангажа.

Три сигнала:

- ***void*** *bindingChangedSignal()* — изменение значения привязки;

- ***void*** *locationSignal (**int** latitude , **int** longitude , **int** height , **bool** isValid)* — значения долготы, широты и высоты;

- ***void*** *orientationSignal (**int** vector , **int** roll , **int** pitch , **bool** isValid)* - значения углов ориентации.

2.2.6 Организация классов AimCalculation, AimCalculationGso и AimCalculationHeo

Для того чтобы произвести наведение на КА, необходимо рассчитать азимут и угол-место. В зависимости от орбиты — свой тип расчёта, поэтому будет два класса, каждый реализует один тип расчёта.

AimCalculationGso реализует расчёт по данным КА ГСО. Свойства делятся на расчётные параметры:

- ***CCoordinate*** *m_latitude* — значение широты АП;

- ***CCoordinate*** *m_longitude* — значение долготы АП;

- ***CCoordinate*** *m_gsoLongitude* — значение долготы КА;

- ***CCoordinate*** *m_longitude* — значение долготы АП.

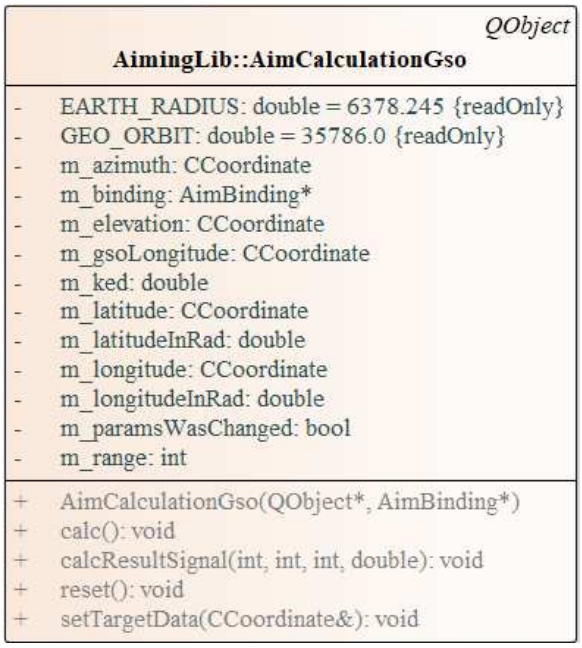

Рисунок 2.12 — UML диаграмма класса AimCalculationGso

А также свойства хранящие результаты расчёта:

- ***CCoordinate*** *m_azimuth* — азимут;
- ***CCoordinate*** *m_elevation* — угол-место;
- ***int*** *m_range* — дальность от АП до КА.

Класс, реализующий расчёт для КА ВЭО — AimCalculationHeo.

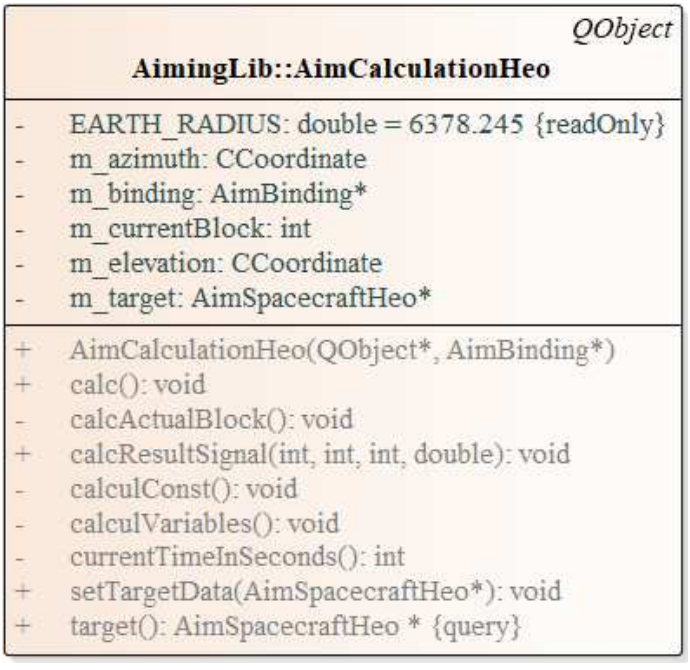

Рисунок 2.13 — UML диаграмма класса AimCalculationHeo

Свойства, содержащие результаты у него такие же как у AimCalculationGso, но расчётные параметры заметно отличаются:

- ***AimSpacecraftHeo*** **m_target* — цель для наведения;
- ***int*** *m_currentBlock* — номер текущего блока с данными.

Единую точку доступа к расчётам будет предоставлять класс AimCalculation. Основные свойства класса AimCalculation:

- ***AimRequest*** **m_request* — указатель на объект запроса на наведение;
- ***bool*** *m_isExecute* — выполняется ли расчёт;
- ***QTimer*** *m_calcTimer* — таймера по срабатыванию которого выполняется расчёт.

Один сигнал:

- ***void*** *aimingSignal(**int** azimuth, **int** elevation)* — расчитанные значения широты и угла-места.

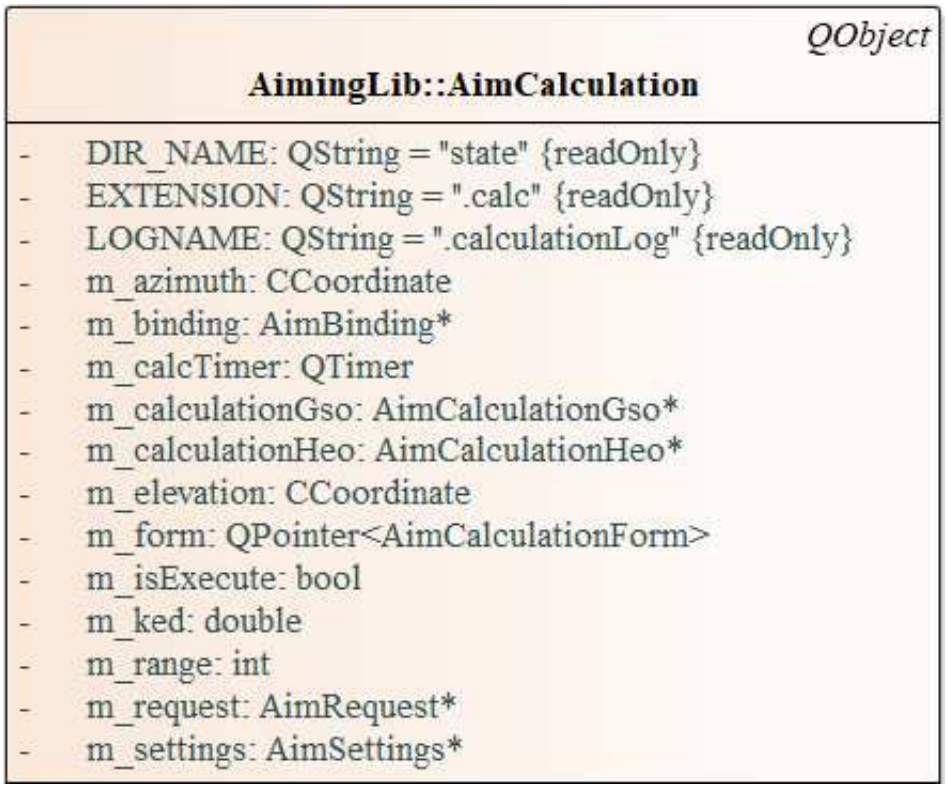

Рисунок 2.14 — UML диаграмма класса AimCalculation

2.2.7 Организация классов AimRequest и AimRequestManager

Необходимо создать такую сущность, через которую пользователь сможет создавать запросы для наведения. Для этого опишем объект AimRequest — реализующий запрос наведения.

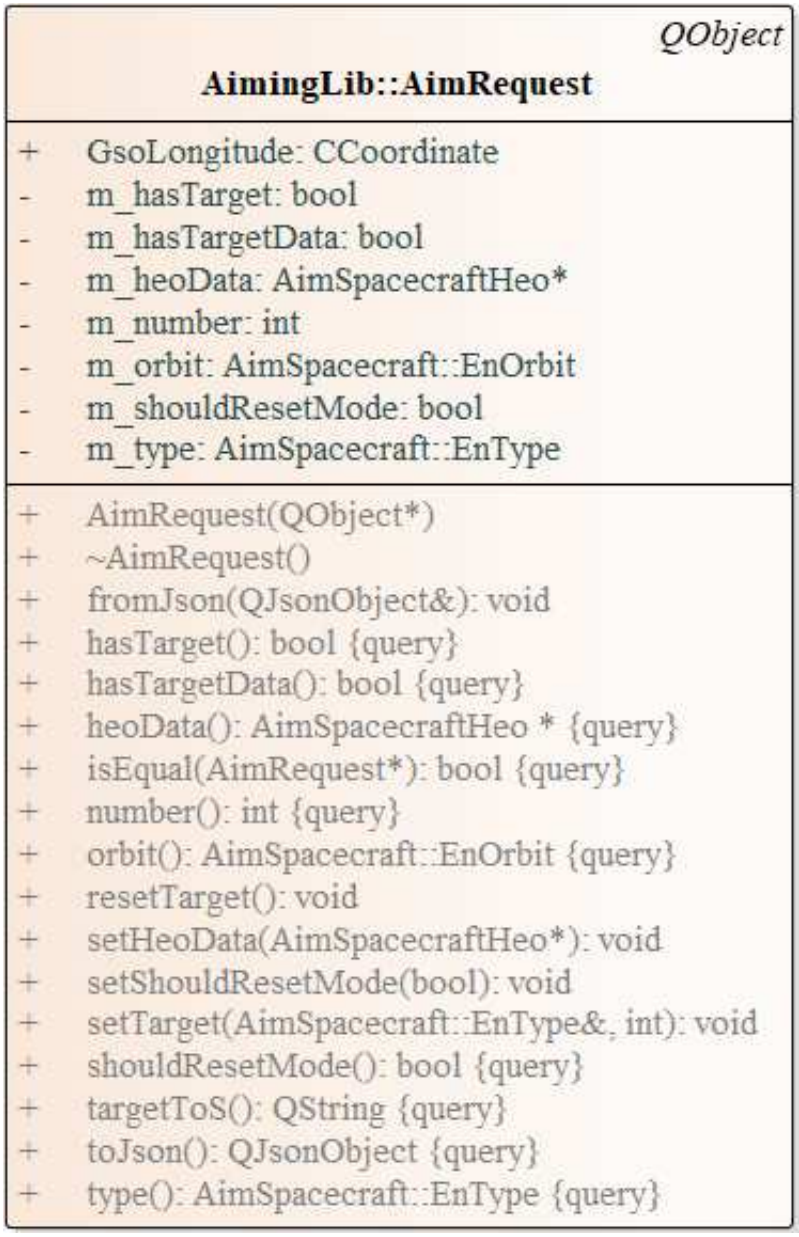

Рисунок 2.15 — UML диаграмма класса AimRequest

Свойства класса AimRequest:

- ***AimSpacecraft*::*EnOrbit*** *m_orbit* — орбита КА на который необходимо выполнить наведение;
- ***AimSpacecraft*::*EnType*** *m_type* — тип КА;

- ***int*** *m_number* — номер КА;
- ***AimSpacecraftHeo*** **m_heoData* — данные для наведения на КА ВЭО;
- ***bool*** *m_hasTarget* — есть ли цель для наведения;
- ***bool*** *m_hasTargetData* — есть ли данные цели для наведения;
- ***bool*** *m_shouldResetMode* — необходимо ли сбросить режим первоначального наведения при сбросе запроса на наведение.

Класс AimRequestManager будет контролировать создание и отправку запросов на наведение.

Рисунок 2.16 — UML диаграмма класса AimRequestManager

Основные свойства класса AimRequestManager:

- ***AimRequest*** **m_currentRequest* — указатель на текущий запрос наведения;

- ***AimRequest*** **m_shosRequest* — СХОС запрос;

- ***bool*** *m_hasDisallowShosRequest* — обрабатывать ли СХОС запросы;

- ***bool*** *m_hasDisallowShowShosDialog* — отображать ли соответствующую форму при получении СХОС запроса;

- ***QTimer*** *m_timer* — таймер проверки получения данных для КА ВЭО.

Два публичных слота:

- ***void*** *shosRequestSlot (**int** type, **int** number, **double** longitude)* — получен новый СХОС запрос;

- ***void*** *resetAimingSlot (**bool** shouldResetMode)* — сброс текущего запроса на наведение.

Один сигнал:

- ***void*** *newRequestSignal(**AimRequest** *request)* — новый запрос на наведение.

2.2.8 Организация классов AimController

Класс AimController будет объединять в себе ранее описанные объекты. Отвечать за создание, удаление и связь этих объектов между собой.

Основные свойства класса AimController:

- ***AimAntenna*** **m_antenna* — указатель на класс антенны;

- ***AimBinding*** **m_binding* — указатель на класс привязки;

- ***AimCalculation*** **m_calculation* — указатель на класс расчётов;

- ***AimSpacecraftsData*** **m_data* — указатель на класс данных КА;

- ***AimRequestManager*** **m_requestManager* — указатель на класс менеджера запросов;

- ***AimSettings*** **m_settings* — указатель на класс настроек.

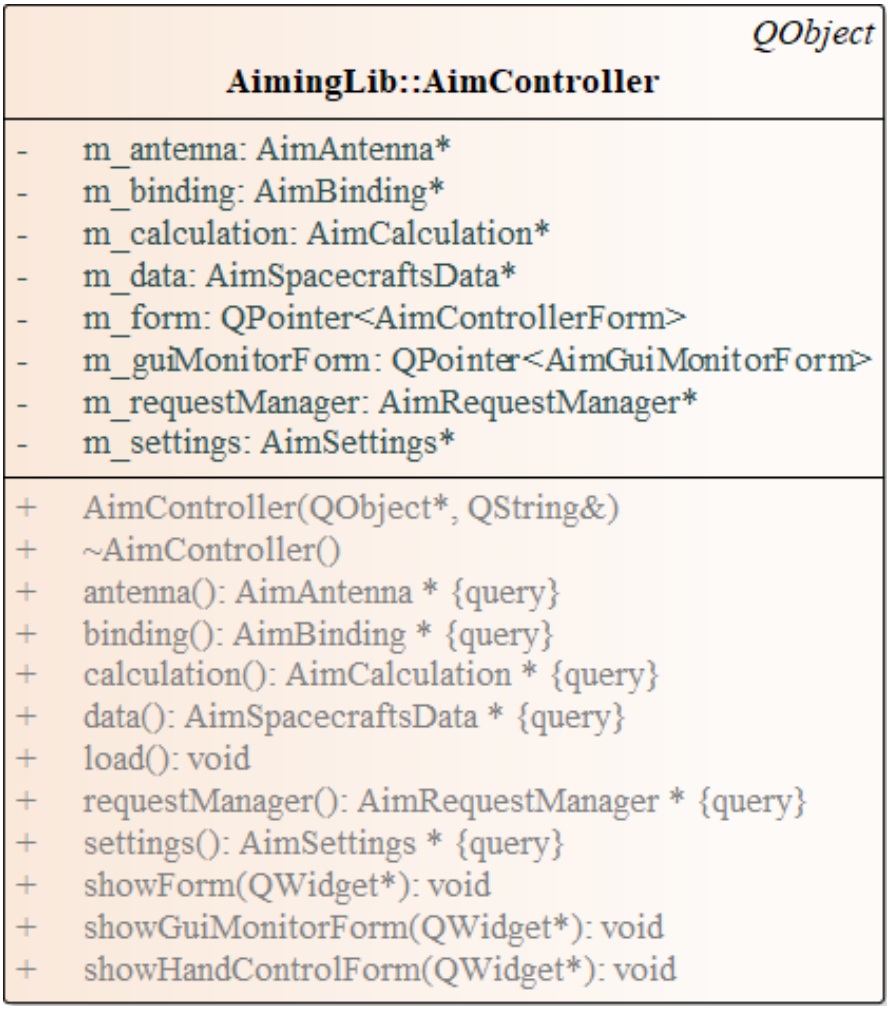

Рисунок 2.17 — UML диаграмма класса AimController

2.2.9 Вывод

Смоделировав и составив UML диаграммы для абстрактной антенны, получилось построить архитектуру по принципу при котором каждый модуль выполняет определённую задачу. Выделились сигналы и слоты, подключение к которым обеспечит связь между абстрактной антенной и реализацией протокола конкретной антенны.

2.3 Модель библиотеки Network

Класс Network будет представлять собой реализацию транспортных протоколов, которые никак несвязанные друг с другом. Такой подход поз-

волит по необходимости подключать конкретный протокол в проект, а в дальнейшем расширять её другими протоколами.

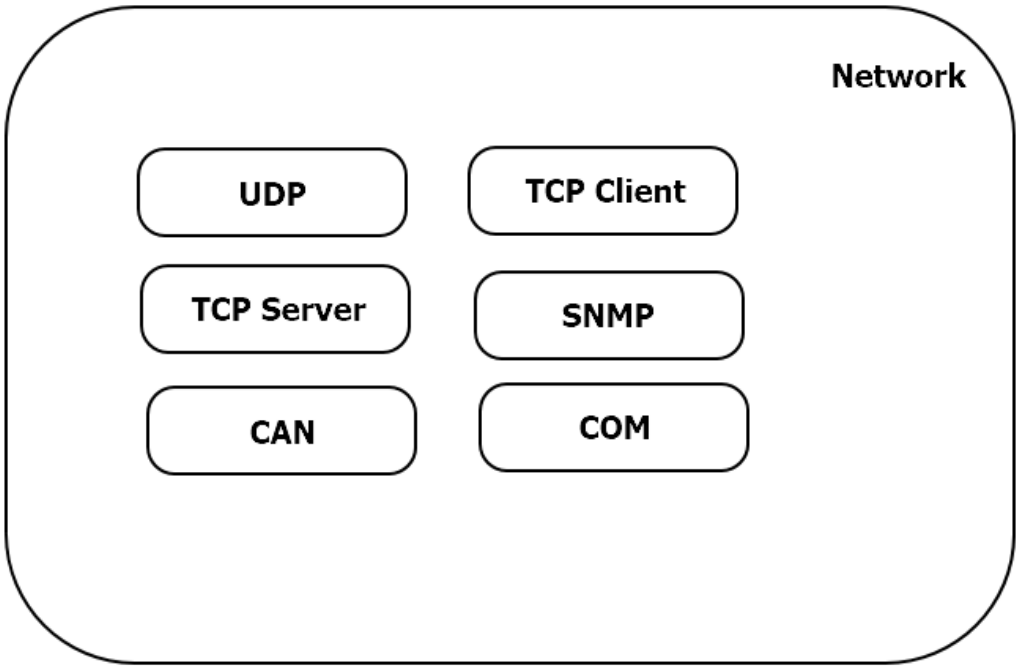

Рисунок 2.18 — Схема проекта Network

2.3.1 Модель UDP Adapter

Для реализации UdpAdapter понадобится два класса. Класс UdpAdapter, в котором будут реализованы механизмы приёма и передачи данных по UDP протоколу. А также сигналы и слоты для связи проекта с другими проектами, которые будут его использовать.

Основные свойства класса UdpAdapter:

- ***QUdpSocket*** **m_socket* — сокет для работы с UDP;
- ***QString*** *m_socketError* — строка для хранения ошибок;
- ***QTimer*** *m_reconnectTimer* — таймер переподключения.

Три слота:

- ***void*** *sendDataSlot (**const QByteArray** &data)* — отправка данных по уже открытому UDP сокету;
- ***void*** *sendDataSlot (**const QString** &ip, **ushort** port, **const QByteArray** &data)* — отправка данных на указанный адрес по UDP сокету;
- ***void*** *readyReadSlot ()* — получение данных по UDP сокету.

Два сигнала:

- ***void*** *receiveDataSignal (****const QByteArray*** *&data)* — отправка полученных данных;

- ***void*** *receiveDataSignal (****const QString*** *&ip,* ***ushort*** *port,* ***const QByteArray*** *&data)* — отправка полученных данных с определённого адреса;

Рисунок 2.19 — UML диаграмма класса UdpAdapter

Класс UdpSettings содержащий настройки необходимые для работы UDP сокета. Основные из них:

- ***QString*** *m_ip* — IP-адрес устройства;
- ***ushort*** *m_port* — UDP-порт устройства;
- ***ushort*** *m_localPort* — собственный UDP-порт.

2.3.2 Модель TcpClient

Реализация TcpClient будет состоять из двух частей. Класс TcpClientAdapter, в котором будут реализованы механизмы приёма и передачи данных по TCP протоколу.

Основные свойства класса TcpClientAdapter:

- ***QTcpSocket*** **m_socket* — сокет для работы с TCP;
- ***QString*** *m_socketError* — строка для хранения ошибок;
- ***QTimer*** *m_reconnectTimer* — таймер переподключения.

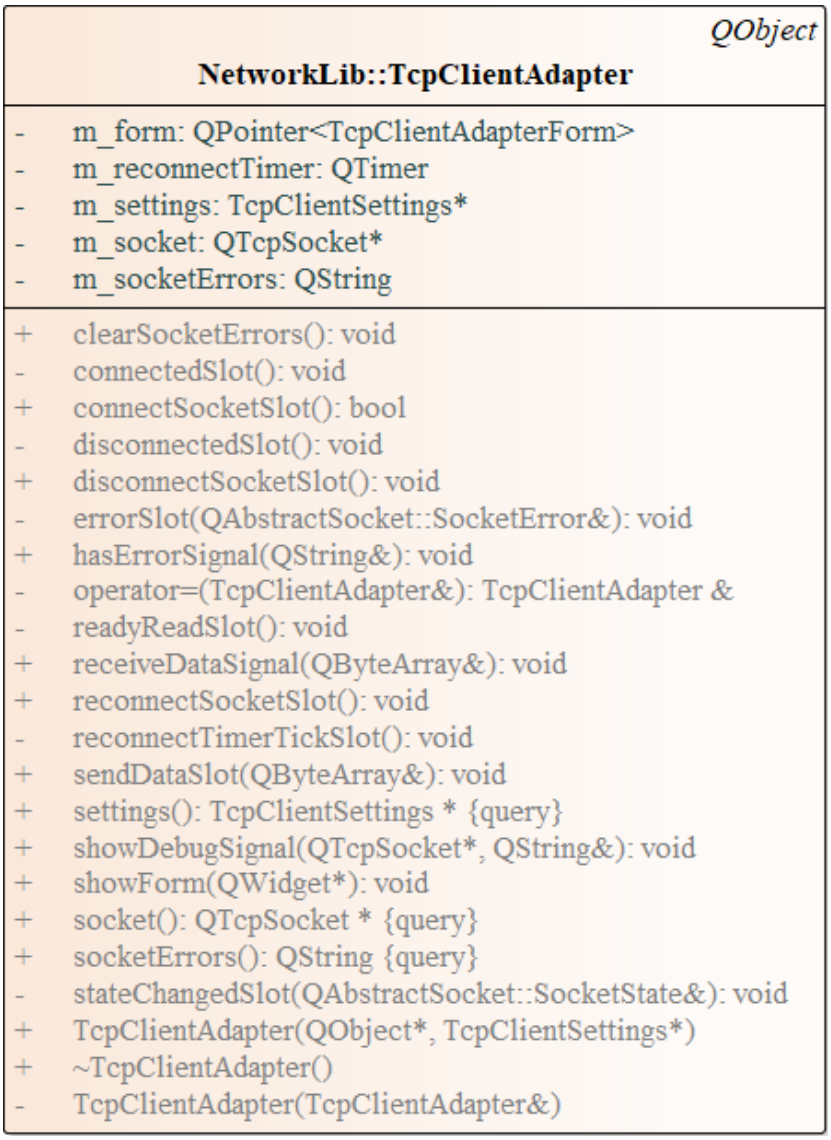

Рисунок 2.20 — UML диаграмма класса TcpClientAdapter

Класс TcpClientSettings содержащий настройки необходимые для работы TCP сокета.

Основные из них:

- ***QString*** *m_ip* — IP-адрес устройства;
- ***ushort*** *m_port* — TCP-порт устройства;
- ***ushort*** *m_localPort* — собственный TCP-порт.

2.3.3 Модель TcpServer

Реализация TcpServer будет состоять из двух частей. В нём будут реализованы механизмы, подключения и отключения клиентов, приёма и передачи данных по TCP протоколу[4].

Класс TcpServerAdapter, который содержит следующие свойства:

- ***QTcpServer*** **m_server* — Qt реализация TCP сервера;
- ***QString*** *m_serverError* — строка для хранения ошибок.

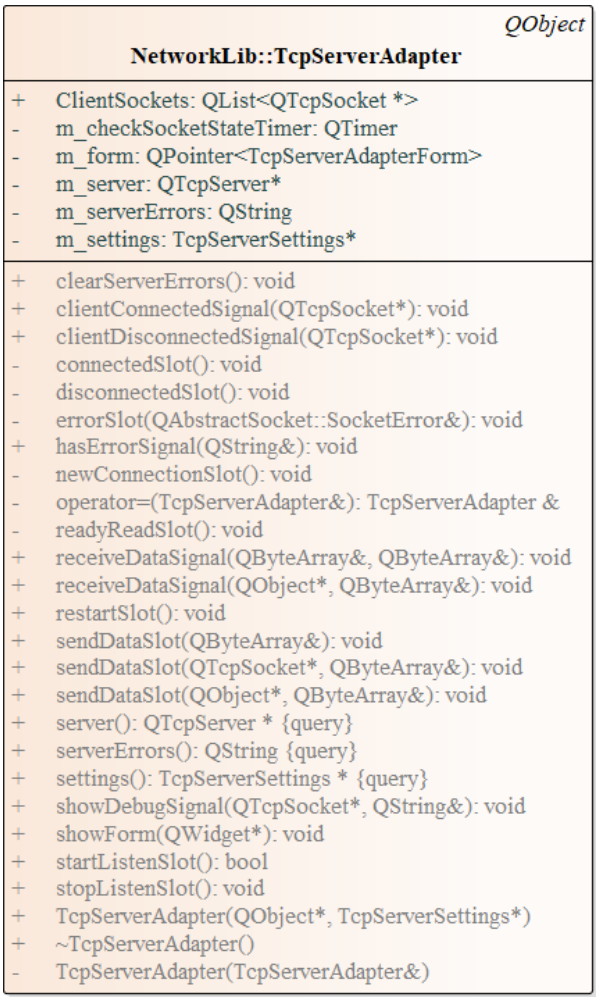

Рисунок 2.21 — UML диаграмма класса TcpServerAdapter

Класс TcpServerSettings содержащий настройки необходимые для работы TCP сокета.

Основные из них:

- ***QString*** *m_ip* — IP-адрес устройства;
- ***ushort*** *m_port* — TCP-порт устройства;
- ***ushort*** *m_localPort* — собственный TCP-порт.

2.3.4 Модель SNMP

В SNMP взаимосвязь клиента с сервером построенна по принципу запрос-ответ[5]. Запрос представляет сообщение определённой формы. Для удобства реализация SNMP будет состоять из трёх частей.

Класс SnmpMessage который содержит параметры SNMP сообщения. Некоторые из основных свойств класса:

- ***int*** *m_id* — идентификатор сообщения;
- ***int*** *m_timeToLive* — время жизни сообщения.

Основные методы класса:

- ***bool*** *fromBytes(**QByteArray** bytes)* — формировка SNMP сообщения из байтов;
- ***QByteArray*** *toBytes()* ***const*** — приведение SNMP сообщения к байтовому представлению.

Класс SnmpAdapter реализующий методы отправки сообщений по Object IDentification (OID) и обработку результатов.

Класс содержит следующие параметры:

- ***UdpAdapter*** **m_udpAdapter* — UDP адаптер;
- ***QList<SnmpMessage*>*** *m_requests* — очередь SNMP сообщений.

Класс SnmpSettings содержащий настройки необходимые для работы SNMP сокета.

Основные из них:

- ***UdpAdapterSettings*** **m_udpSettings* — настройки UDP сокета;
- ***QString*** *m_readCommunity* — пароль на чтение;
- ***QString*** *m_writeCommunity* — пароль на запись;
- ***int*** *m_timeToLive* — максимальное время жизни SNMP сообщений.

Рисунок 2.22 — UML диаграмма класса SnmpAdapter

2.3.5 Модель COM протокола

Реализация COM будет состоять из двух частей.

Класс ComAdapter осуществляющий создание и удаление COM порта, а также организующий приём и отправку данных. Основные свойства ComAdapter:

- ***QSerialPort** * m_serialPort* — указатель на объект COM порта;
- ***bool** m_isPortCreated* — создан ли COM порт;
- ***QString** m_portErrors* — ошибки порта;
- ***QTimer** m_reconnectTimer* — таймер переподключения.

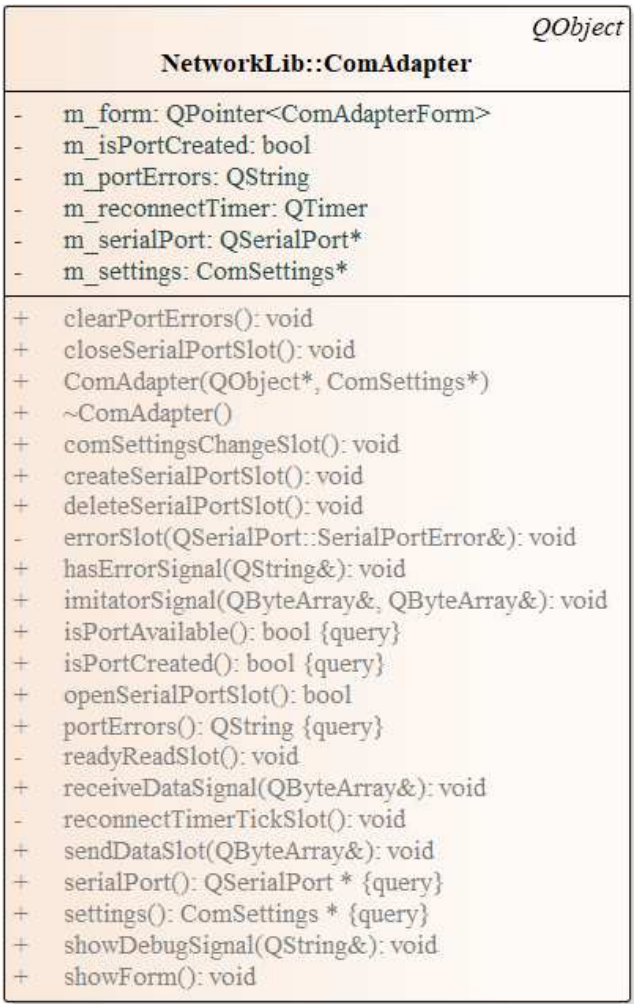

Рисунок 2.23 — UML диаграмма класса ComAdapter

Класс ComSettings содержащий настройки необходимые для работы SNMP сокета. Такие как:

- ***QString*** *m_comPort* — имя COM порта;
- ***int*** *m_baudRates* — скорость передачи;
- ***int*** *m_dataBits* — кол-во бит с данными;
- ***int*** *m_parity* — чётность;
- ***int*** *m_stopBits* — количество стоп-битов;
- ***int*** *m_flowControl* — управление потоком;
- ***QTimer*** *m_reconnectTimer* — таймер переподключения.

2.3.6 Модель CAN протокола

Реализация CAN будет состоять из трёх частей.

CanProtocolMessage описывает один перечисляемый тип:

- ***enum EnType*** — возможный тип CAN сообщения.

Основные свойства класса:

- ***int*** *m_id* — идентификатор фрейма;
- ***QByteArray*** *m_data* — поле данных сообщения;
- ***EnType*** *m_type* — тип сообщения;
- *uchar m_command* — код команды;
- ***int*** *m_ttl* — время жизни сообщения;
- ***bool*** *m_shouldChecked* — необходимость проверки сообщения.

Основные методы класса:

- ***QByteArray*** *toTcpFormat()* ***const*** — приводит CAN сообщение к сообщению TCP формата;
- ***QByteArray*** *toRawFormat (****const CanProtocolSettings::EnCanVersion*** *&version)* ***const*** — в зависимости от версии преобразует поле данных в байты.

В классе CanProtocol будет возможность организовать работу протокола на одном из двух адаптерах: TCP или COM.

Класс содержит следующие параметры:

- ***TcpClientAdapter*** **m_tcpAdapter* — реализация TCP адаптера;
- ***TcpServerAdapter*** **m_tcpServer* — реализация TCP сервера;
- ***ComAdapter*** **m_comAdapter* — COM адаптер;
- ***QByteArray*** *m_buffer* — буфер для входящих данных;
- ***QTimer*** *m_checkBufferTimer* — таймер проверки буфера входящих данных;
- ***QList<int>*** *m_frameIdList* — список идентификаторов фреймов на которых подписывается клиент.

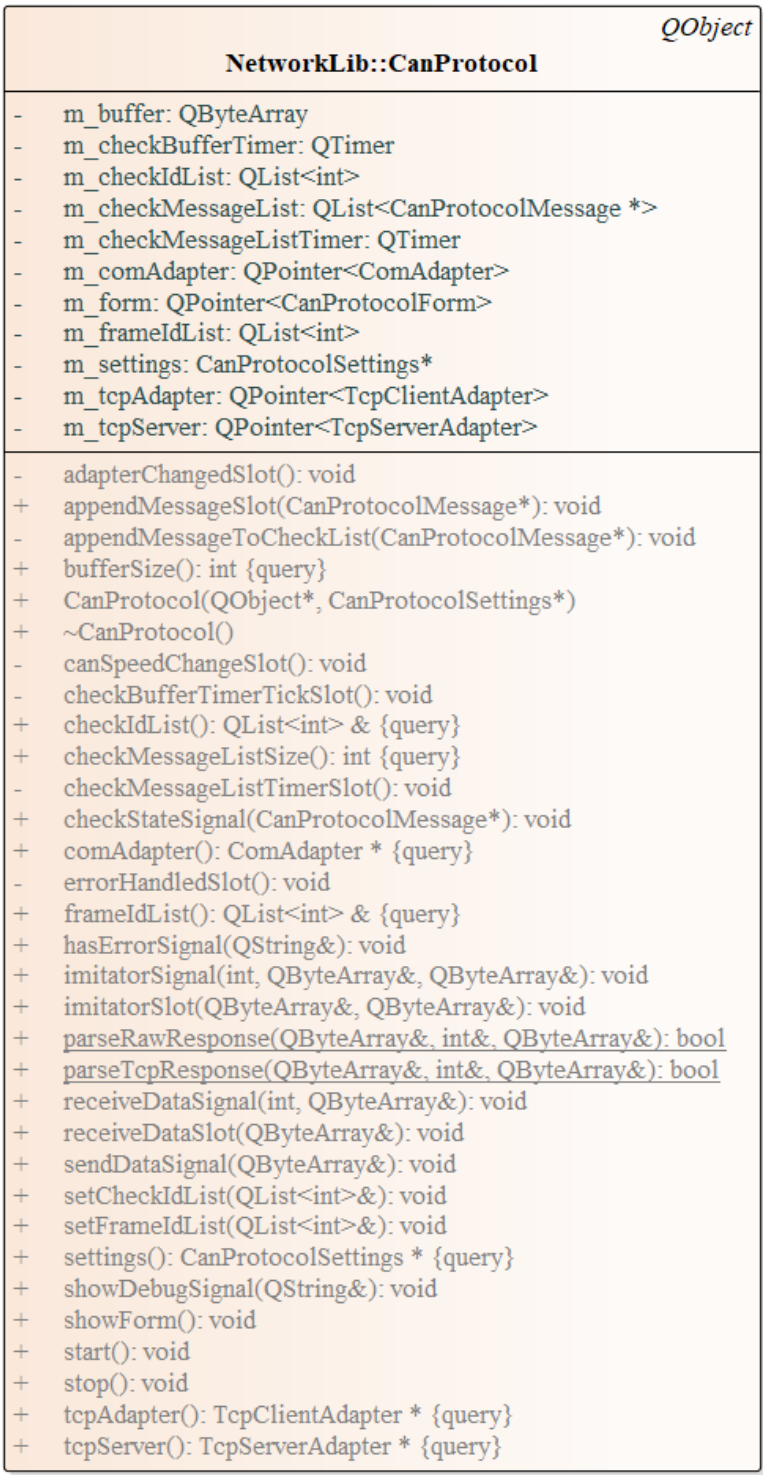

Рисунок 2.24 — UML диаграмма класса CanAdapter

Класс CanProtocolSettings содержащий настройки необходимые для работы CAN протокола.

- ***CanProtocolSettings*** описывает четыре перечисляемых типа:
- ***enum EnAdapterType*** — возможные типы адаптера;
- ***enum EnCanSpeed*** — возможные скорости работы CAN протокола;

- ***enum EnCanType*** — тип CAN;
- ***enum EnCanVersion*** — версия CAN.

Основные свойства класса CanProtocolSettings:

- ***EnAdapterType*** *m_adapterType* — текущий апатер CAN протокола;
- ***EnCanType*** *m_canType* — текущий тип CAN протокола;
- ***EnCanSpeed*** *m_canSpeed* — текущая скорость CAN протокола;
- ***EnCanVersion*** *m_canVersion* — текущая версия CAN протокола;
- ***ushort*** *m_checkMessageListInterval* — интервал проверки списка сообщений.

2.3.7 Вывод

Смоделировав и составив UML диаграммы для проекта Network, получилось построить архитектуру по принципу никак несвязанных между собой реализаций протоколов UDP, TCP, SNMP, COM и CAN. Также были предусмотрены сигналы и слоты с помощью которых будет производиться подключение к проектам.

3 Разработка системы

Для реализации разработанных модулей, было создано два отдельных проекта: Aiming и Network.

3.1 Реализация Aiming

По смоделированным выше классам был разработан проект Aiming рассмотрим основные моменты принципы разработки.

3.1.1 Реализация AimController

В конструкторе класса AimController происходит создание модулей, а также соединений их сигналов и слотов.

```
AimController::AimController(QObject *parent, const QString &systemName) :
        QObject(parent)
{
        m_settings = new AimSettings(this, systemName);
        m_settings->debugOut("AimController | AimController was created.")
            ;
        m_data = new AimSpacecraftsData(this, m_settings);
        m_antenna = new AimAntenna(this, m_settings);
        m_binding = new AimBinding(this, m_settings);
        m_calculation = new AimCalculation(this, m_settings, m_binding);
        connect(m_calculation, &AimCalculation::aimingSignal,
                m_antenna, &AimAntenna::aimingSlot);
        m_requestManager = new AimRequestManager(this, m_settings, m_data)
            ;
        connect(m_requestManager, &AimRequestManager::newRequestSignal,
                m_calculation, &AimCalculation::newRequestSlot);
        connect(m_requestManager, &AimRequestManager::newRequestSignal,
                m_antenna, &AimAntenna::newRequestSlot);
        connect(m_antenna, &AimAntenna::resetAimingSignal,
                m_requestManager, &AimRequestManager::resetAimingSlot);
}
```

В деструкторе класса происходит очистка памяти. А также удаление graphical user interface (GUI) форм, если они были созданы.

```
AimController::~AimController()
{
```

```
        if (m_form != nullptr)
                delete m_form;
        if (m_guiMonitorForm != nullptr)
                delete m_guiMonitorForm;
        delete m_requestManager;
        delete m_calculation;
        delete m_binding;
        delete m_data;
        delete m_antenna;
        m_settings->debugOut("AimController | AimController was deleted.")
            ;
        delete m_settings;
}
```

Для каждого модуля проекта Aiming в классе AimController реализован get-метод позволяющий проектам, которые будут использовать Aiming, подключиться напрямую к модулям. Пример такого get-метода приведён ниже.

```
AimAntenna *AimController::antenna() const
{
        return m_antenna;
}
```

3.1.2 Реализация сигналов и слотов на примере AimAntenna

Для получения от антенны текущего состояния был спроектирован публичный слот *void stateModeSlot(int pointingMode, int correctionMode, bool isOnline)*. Его реализация приведена ниже. В нём отправляется сигнал при условии, что в настройках задан необходимый параметр.

```
void AimAntenna::stateModeSlot(int pointingMode, int correctionMode, bool
    isOnline)
{
        EnPointingMode receivedPointingMode = static_cast<EnPointingMode>(
            pointingMode);
        EnCorrectionMode receivedCorrectionMode = static_cast<
            EnCorrectionMode>(correctionMode);
        if (m_settings->shouldCheckAntennaRegime())
        {
                if (m_pointingMode != receivedPointingMode)
                {
                        emit pointingModeSignal(static_cast<int>(
                            m_pointingMode));
                }
                if (m_correctionMode != receivedCorrectionMode)
```

```
                {
                        emit correctionModeSignal(static_cast<int>(
                            m_correctionMode));
                }
        }
        else
        {
                if (shouldResetAiming(receivedPointingMode) && isOnline)
                        resetAiming(false);
                m_pointingMode = receivedPointingMode;
                m_correctionMode = receivedCorrectionMode;
        }
}
```

Также пример отправки сигнала можно увидеть в приватном слоте *void aimingSlot(int azimuth, int elevation)*, он срабатывает при получение рассчитанных целеуказаний от модуля AimCalculation. При получении данных происходит отправка сигнала с целеуказаниями.

3.1.3 Пример подключения проекта Aiming

Для примера подключения проекта Aiming необходимо рассмотреть конечный проект станции - Acs. В классе AcsController подключается библиотека Aiming, а также библиотека с реализацией протокола обмена конкретной БУА. В конструкторе класса вызывается метод *void createEntities()*, в котором создаётся AimController. После создания AimController, вызываются методы настройки.

```
void AcsController::createEntities()
{
        m_aiming = new AimController(nullptr, "aiming");
        m_aiming->settings()->setShouldShowDebugInfo(false);
        m_aiming->settings()->setShouldShowCalculation(false);
        m_aiming->settings()->setCanHandleRequestByShos(false);
        m_aiming->settings()->setHasHeo(false);
        m_aiming->settings()->setHasAutosearching(false);
        m_aiming->settings()->setShouldCheckAntennaRegime(false);
        ...
}
```

Далее в конструкторе вызывается метод *void createDevices()*, в котором происходит создание классов устройств входящих в станцию. Ниже представлен часть метода *void createDevices()*, в котором создаётся объект блока

управления антеннами (БУА).

```
void AcsController::createDevices()
{
        ...

        m_bua = new BuaLivenLib::BuaController(nullptr, "bua");
        m_bua->settings()->setName("AFU 2,5B-BM");
        m_bua->settings()->canProtocolSettings()->setAdapterType(
            CanProtocolSettings::EnAdapterType::TCP);
        m_bua->settings()->setIp("192.168.1.126");
        m_bua->settings()->setPort(2001);
        m_bua->settings()->apply();

        ...
}
```

Далее в конструкторе вызывается метод *void connectAimingWithAntenna(AimController *aiming, BuaLivenLib::BuaController *antenna)*. В нём происходит соединений сигналов и слотов двух проектов.

```
void AcsController::connectAimingWithAntenna(AimController *aiming,
        BuaLivenLib::BuaController *antenna)
{
        connect(aiming->antenna(), SIGNAL(requestStateViewSignal(QWidget
            *, QWidget *)),
                antenna, SLOT(requestStateViewSlot(QWidget *, QWidget *)))
                    ;
        connect(aiming->antenna(), SIGNAL(requestRegimeViewSignal(QWidget
            *)),
                antenna, SLOT(requestRegimeViewSlot(QWidget *)));

        connect(aiming->antenna(), SIGNAL(resetSignal()),
                antenna, SLOT(resetSlot()));
        connect(aiming->antenna(), SIGNAL(rotateSignal(int)),
                antenna, SLOT(rotateSlot(int)));
        connect(aiming->antenna(), SIGNAL(trimSignal(int)),
                antenna, SLOT(trimSlot(int)));

        connect(aiming->antenna(), SIGNAL(designationSignal(int, int)),
                antenna, SLOT(designationSlot(int, int)));

        connect(aiming->antenna(), SIGNAL(pointingModeSignal(int)),
                antenna, SLOT(pointingModeSlot(int)));
        connect(aiming->antenna(), SIGNAL(correctionModeSignal(int)),
                antenna, SLOT(correctionModeSlot(int)));

        connect(aiming->binding(), SIGNAL(orientationSignal(int, int, int,
            bool)),
```

```
            antenna, SLOT(orientationSlot(int, int, int, bool)));
        connect(aiming->binding(), SIGNAL(locationSignal(int, int, int,
            bool)),
            antenna, SLOT(locationSlot(int, int, int, bool)));

        connect(aiming->antenna(), SIGNAL(signalLevelSignal(int, double)),
            antenna, SLOT(signalLevelSlot(int, double)));
        connect(antenna, SIGNAL(statePointingModeSignal(int)),
            aiming->antenna(), SLOT(statePointingModeSlot(int)));
        connect(antenna, SIGNAL(stateCorrectionModeSignal(int)),
            aiming->antenna(), SLOT(stateCorrectionModeSlot(int)));
        connect(antenna, SIGNAL(stateStopTrackersSignal(int, int, int, int
            )),
            aiming->antenna(), SLOT(stateStopTrackersSlot(int, int,
                int, int)));
}
```

3.2 Реализация Network

Так как реализации протоколов не связанны между собой и их реализация похожа, достаточно будет рассмотреть один протокол.

3.2.1 Реализация Network на примере UdpAdapter

В конструкторе класса UdpAdapter происходит создание UDP сокета и соединение сигналов и слотов.

```
UdpAdapter::UdpAdapter(QObject *parent, UdpAdapterSettings *settings) :
        QObject(parent),
        m_settings(settings)
{
        connect(m_settings, &UdpAdapterSettings::reconnectSocketSignal,
            this, &UdpAdapter::reconnectSocketSlot);
        m_socket = new QUdpSocket(this);
        connect(m_socket, &QUdpSocket::readyRead, this, &UdpAdapter::
            readyReadSlot);
        connect(m_socket, static_cast<void (QUdpSocket::*)(QAbstractSocket
            ::SocketError)>
            (&QAbstractSocket::error), this, &UdpAdapter::errorSlot);
        connect(m_socket, static_cast<void (QUdpSocket::*)(QAbstractSocket
            ::SocketState)>
            (&QAbstractSocket::stateChanged), this, &UdpAdapter::
                stateChangedSlot);
```

```
        connect(&m_reconnectTimer, &QTimer::timeout, this, &UdpAdapter::
            reconnectTimerTickSlot);
        m_reconnectTimer.start(m_settings->reconnectInterval());
        for (auto host : QNetworkInterface::allAddresses())
        {
                if (host.protocol() == QAbstractSocket::IPv4Protocol)
                        m_localIpList.append(NetworkRepo::toIpString(host)
                            );
        }
        m_settings->debugOut("Created");
}
```

Слот *void sendDataSlot(const QString &ip, ushort port, const QByteArray &data)* используется для отправки данных по UDP сокету.

```
void UdpAdapter::sendDataSlot(const QString &ip, ushort port, const
    QByteArray &data)
{
        if (m_socket->state() != QAbstractSocket::BoundState)
                return;
        QHostAddress host = QHostAddress(ip);
        m_socket->writeDatagram(data, host, port);

        QString suffix = QString("{%1:%2}")
                .arg(ip)
                .arg(port);
        QString debugString = NetworkRepo::buildOutMsg(data, m_settings->
            isImitator(), suffix);
        emit showDebugSignal(debugString);
        m_settings->debugOut(debugString);
}
```

Сигнал *void receiveDataSignal(const QByteArray &data)* вызывается при получение данных на UDP сокет в слоте *void readyReadSlot()*.

```
void UdpAdapter::readyReadSlot()
{
        ...
        emit receiveDataSignal(senderIp, m_settings->localPort(), datagram
            );
}
```

3.2.2 Пример подключения проекта Network

Для примера необходимо разобрать проект реализующий протокол обмена для АП Ka/Q-диапозон. А конкретно класс BuaController в котором про-

исходит подключение библиотеки Network. Класс BuaController наследуется от класса QThread, для обеспечения многопоточности. Запуск отдельного потока происходит в переопределённом методе *void run()*. В нём же создаётся и подключается класс UdpAdapter.

```
void BuaController::run()
{
        m_protocol = new BuaProtocol(nullptr, m_settings, m_device);
        connect(this, &QThread::finished, m_protocol.data(), &QObject::
            deleteLater);
        connect(this, &QThread::finished, m_protocol.data(), &QObject::
            deleteLater);
        connect(m_protocol.data(), &BuaProtocol::handleErrorSignal,
                this, &BuaController::handleErrorSlot);
        connect(this, &BuaController::errorHandledSignal,
                m_protocol.data(), &BuaProtocol::errorHandledSlot);
        connect(m_device, &BuaModel::appendMessageSignal,
                m_protocol.data(), &BuaProtocol::appendMessageSlot);
        m_udpAdapter = new UdpAdapter(nullptr, m_settings->
            udpAdapterSettings());
        connect(this, &QThread::finished, m_udpAdapter.data(), &QObject::
            deleteLater);
        m_udpAdapter->connectSocketSlot();
        connect(m_protocol.data(), &BuaProtocol::sendDataSignal,
                m_udpAdapter.data(), &UdpAdapter::sendDataSlot);
        connect(m_udpAdapter.data(), &UdpAdapter::receiveDataSignal,
                m_protocol.data(), &BuaProtocol::receiveDataSlot);
        connect(m_udpAdapter.data(), &UdpAdapter::imitatorSignal,
                m_protocol.data(), &BuaProtocol::imitatorSlot,
                Qt::DirectConnection);
        m_device->setEnable(true);
        exec();
        m_device->moveToThread(qApp->thread());

        m_device->setOnline(false);
        m_device->setEnable(false);
}
```

3.3 Вывод

Был реализован и рассмотрен вариант подключения библиотеки управления Aiming. Представляющий из себя абстрактную антенну, с помощью которой можно управлять антенной. Также разработан класс Network реализующий сетевые протоколы. Его можно использовать в других проектах.

4 Обзор разработанной системы

Комплексная информация о состоянии оборудования сводится в единую область – мнемосхему, отображающую текущую картину функционирования оборудования. Реализована возможность оперативного переключения между различными АП, не требующего остановки или перезагрузки программного обеспечения подсистемы автоматизированного управления (ПАУ). Графический интерфейс оператора представлен на рисунке 4.1.

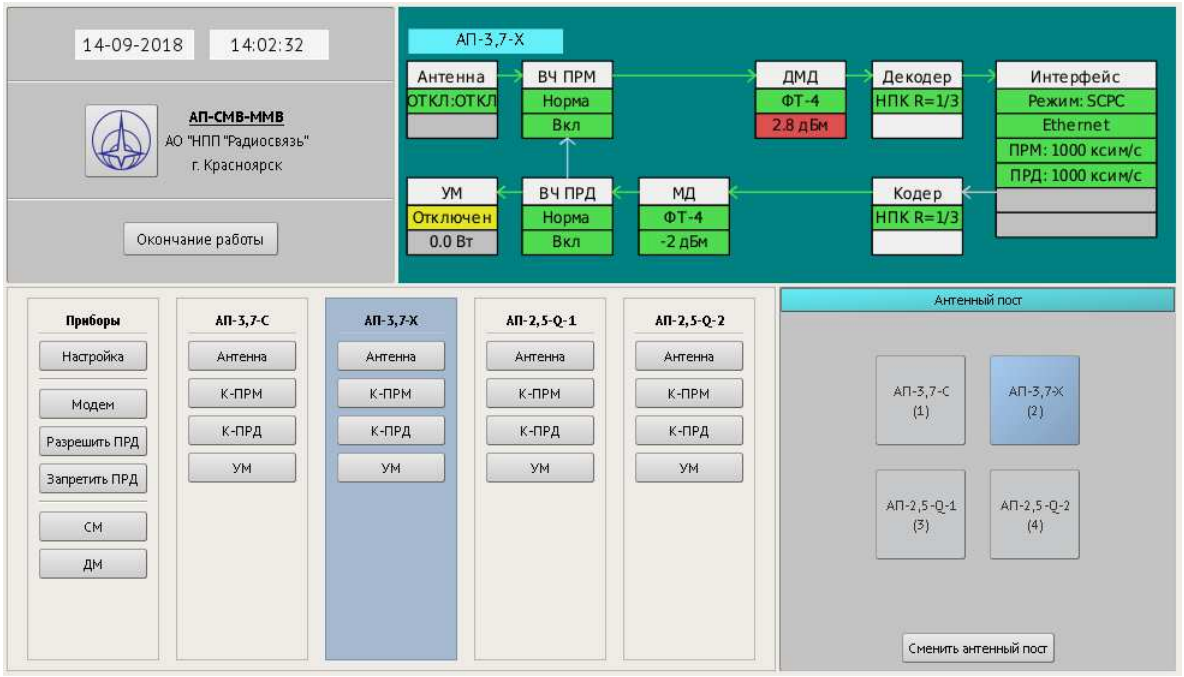

Рисунок 4.1 — Графический интерфейс оператора ПАУ

4.1 Окно управления антенной

В АП представлены антенны, которые различны как по командам управления, так и по протоколам информационно-логической совместимости, для решения этой задачи были проанализированы и выделены обобщенные характеристики и функции антенн, на основе которых был разработан унифицированный программный интерфейс управления, который представлен на рисунке 4.2.

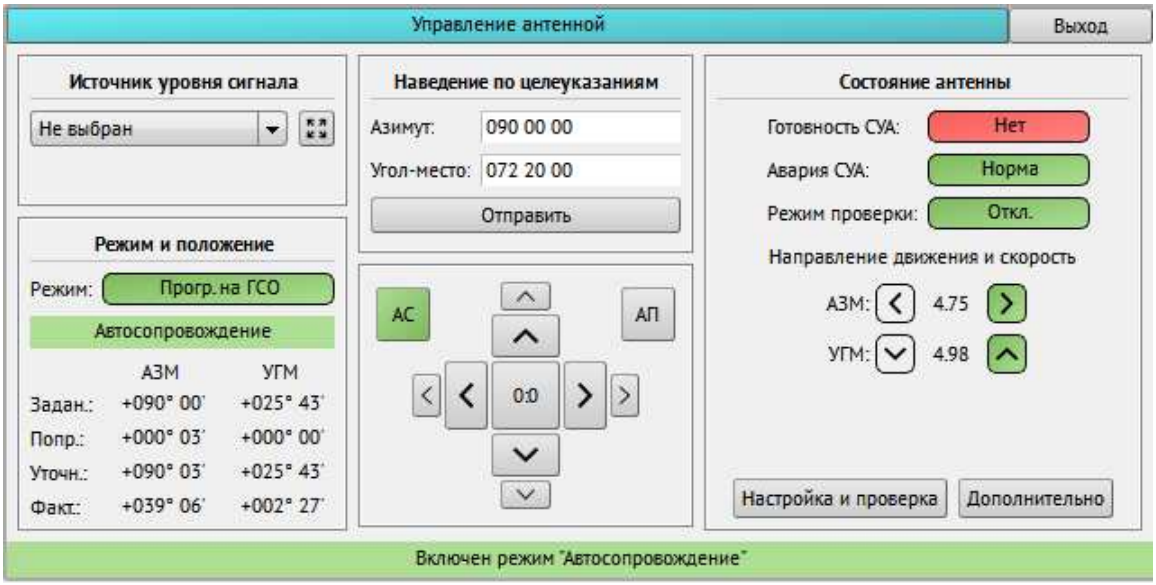

Рисунок 4.2 — Интерфейс управления антенной

Разработка ПО системы управления протекает параллельно с разработкой и изготовлением аппаратуры, и к моменту готовности аппаратного обеспечения ПО уже должно выполнять большую часть своего функционала. Поэтому для тестирования алгоритмов работы и комплексной отладки программных модулей созданы имитаторы устройств входящих в состав АП: конвертеров, усилителей мощности, БУА.

Применение при разработке ПО ПАУ программных имитаторов позволяет в достаточно полной мере реализовать необходимый функционал, и выполнить тестирование основных алгоритмов управления как отдельными приборами, так и комплексом аппаратуры в целом. Совершенствование и расширение возможностей программных имитаторов предоставляет возможность дополнительного тестирования ПО ПАУ, которое включает моделирование временных задержек при работе с оборудованием, моделирование неисправностей и особых состояния, требующих специальных процедур обработки.

4.2 Вывод

Разработанный класс абстрактной антенны полностью справляется со своими задачами и при необходимости имеет возможность к расширению своих возможностей, по средству реализации необходимых сетевых прото-

колов. Этими качествами обладают и имитаторы других устройств входящих в состав АП.

ЗАКЛЮЧЕНИЕ

В процессе выполнения работы были исследованы антенные посты, выделен общий функционал и выбран оптимальный способ решения задачи. Пояснительная записка к выполненной работе содержит информацию о применённых технологиях и разработанной архитектуре программной системы унифицированный метода управления антенными постами контрольно-измерительного комплекса космических аппаратов. Использование механизма сигнал-слот позволило добиться необходимого уровня абстракции в проекте.

Реализована библиотека Aiming, которая при подключение к проектам АП, обеспечивает доступ к основному функционалу антенны. Такой подход позволил снизить трудозатраты и увеличить эффективность процесса разработки проектов, в которых присутствуют антенны, в том числе КИНК. Также была реализована библиотека Network. В ней реализованы такие протоколы, как TCP, UDP, COM и SNMP. Использование библиотеки Network также позволяет снизить трудозатраты при разработке проектов, в которых есть сетевое взаимодействие. В дальнейшем, при необходимости библиотека будет расширяться другими протоколами.

Для тестирования библиотек были реализованы имитаторы устройств входящих в состав АП. Разработанный класс абстрактной антенны полностью справляется со своими задачами и при необходимости имеет возможность к расширению. Библиотека Network в текущем ввиде может использоваться в других проектах. Также при необходимости её можно расширить.

СПИСОК СОКРАЩЕНИЙ

ИСЗ — Искусственный спутник земли

КИНК — Контрольно-измерительный наземный комплекс

БРТК — Бортовой ретрансляционный комплекс

КА — Космический аппарат

КПА — Контрольно-проверочная аппаратура

АП — Антенный пост

ПО — Программное обеспечение

СХОС — Схема организации связи

ГСО — Геостационарная орбита

ВЭО — Высокая эллиптическая орбита

ЦУ — Целеуказания

OID — Object Identification

GUI — Graphical user interface

ПАУ — Подсистема автоматизированного управления

БУА — Блок управления антенной

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Александров С. Советские спутники и космические корабли // Omega Science. — 1961. — С. 440.

2. Мир спутников: что, где и чье на околоземной орбите? [Электронный ресурс] // Журнал «ВВС». — Режим доступа: https://oko-planet.su/science/sciencecosmos/65487-mir-sputnikov-chto-gde-i-che-na-okolozemnoy-orbite.html (дата обращения: 2019-05-23).

3. Шлее М. Qt 5.10. Профессиональное программирование на C++ // Издательство БХВ-Петербург. — 2018. — С. 1072.

4. Таненбаум Э. Компьютерные сети, 5-е издание // Издательский дом "Питер". — 2012. — С. 960.

5. Олифер В. Компьютерные сети. Принципы, технологии, протоколы. Учебник // Издательский дом "Питер". — 2017. — С. 992.

6. СТО 4.2–07–2014 Система менеджмента качества. Общие требования к построению изложению и оформлению документов учебной деятельности. — Введ. 2014. – Красноярск : СФУ, 2014. – 60 с.

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Кафедра «Вычислительная техника»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись
«03» 07 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Унифицированный метод управления антенными постами
контрольно-измерительного комплекса космических аппаратов

09.04.01 — Информатика и вычислительная техника

09.04.01.04 — Технологии разработки программного обеспечения

Научный руководитель	_____ подпись, дата 01.07.19	вед. инженер, канд.техн.наук	В.А. Хабаров
Выпускник	_____ подпись, дата 01.07.19		Д.В. Сафиуллин
Рецензент	_____ подпись, дата 01.07.19	вед. инженер, аспирант	И.Н. Рыженко
Нормоконтролер	_____ подпись, дата 02.07.19	доцент, канд.техн.наук	А.И. Постников

Красноярск 2019